Tutorial Summary

Christopher Plumberg

Lund University

August 2, 2019

Test-driven development (TDD)

The main idea: TDD is a general approach to developing testable, safe, modular code *via* unit tests.

The main algorithm:

Red -

Green -

Refactor -

Test-driven development (TDD)

The main idea: TDD is a general approach to developing testable, safe, modular code *via* unit tests.

The main algorithm:

Red - Write a minimal failing test

Green -

Refactor -

Test-driven development (TDD)

The main idea: TDD is a general approach to developing testable, safe, modular code *via* unit tests.

The main algorithm:

Red - Write a minimal failing test

Green - Pass test with minimal change

Refactor -

Test-driven development (TDD)

The main idea: TDD is a general approach to developing testable, safe, modular code *via* unit tests.

The main algorithm:

Red - Write a minimal failing test

Green - Pass test with minimal change

Refactor - Clean up test/production code

The initial function:

```
std::string arabic_to_roman(int arabicNumber)
{
  return "";
}
```

Red - Write minimal failing test:

```
TEST(RomanTest, TestNegativeNumber) {
        EXPECT_EQ(arabic_to_roman(-1), "zzz");
}
```

Red - Write minimal failing test:

```
TEST(RomanTest, TestNegativeNumber) {
        EXPECT_EQ(arabic_to_roman(-1), "zzz");
}
```

Running this code results in failure!

Green - Pass test with minimal change
The updated function:

```
std::string arabic_to_roman(int arabicNumber)
{
  if (arabicNumber <= 0) return "zzz";
  return "";
}
```

Green - Pass test with minimal change
The updated function:

```
std::string arabic_to_roman(int arabicNumber)
{
  if (arabicNumber <= 0) return "zzz";
  return "";
}
```

Now running the test results in success!

Refactor - Clean up test/production code

The tested function:

```
std::string arabic_to_roman(int arabicNumber)
{
  // Make sure number is positive.
  if ( arabicNumber <= 0 ) return "zzz";

  std::string result = "";

  return ( result );
}
```

Refactor - Clean up test/production code

The tested function:

```
std::string arabic_to_roman(int arabicNumber)
{
  // Make sure number is positive.
  if ( arabicNumber <= 0 ) return "zzz";

  std::string result = "";

  return ( result );
}
```

Running the test (still) results in success and the code's function is clear!

And repeat *ad infinitum*.

```
TEST(RomanTest, TestNegativeNumber) {
        EXPECT_EQ(arabic_to_roman(-1), "zzz");
}

TEST(RomanTest, TestOnesConversion) {
        EXPECT_EQ(arabic_to_roman(1), "I");
        EXPECT_EQ(arabic_to_roman(2), "II");
        EXPECT_EQ(arabic_to_roman(3), "III");
        EXPECT_EQ(arabic_to_roman(4), "IV");
        EXPECT_EQ(arabic_to_roman(5), "V");
        EXPECT_EQ(arabic_to_roman(6), "VI");
        EXPECT_EQ(arabic_to_roman(7), "VII");
        EXPECT_EQ(arabic_to_roman(8), "VIII");
        EXPECT_EQ(arabic_to_roman(9), "IX");
}
```

```
TEST(RomanTest, TestNegativeNumber) {
        EXPECT_EQ(arabic_to_roman(-1), "zzz");
}

TEST(RomanTest, TestOnesConversion) {
        EXPECT_EQ(arabic_to_roman(1), "I");
        EXPECT_EQ(arabic_to_roman(2), "II");
        EXPECT_EQ(arabic_to_roman(3), "III");
        EXPECT_EQ(arabic_to_roman(4), "IV");
        EXPECT_EQ(arabic_to_roman(5), "V");
        EXPECT_EQ(arabic_to_roman(6), "VI");
        EXPECT_EQ(arabic_to_roman(7), "VII");
        EXPECT_EQ(arabic_to_roman(8), "VIII");
        EXPECT_EQ(arabic_to_roman(9), "IX");
}

TEST(RomanTest, TestTensConversion) {
        EXPECT_EQ(arabic_to_roman(10), "X");
        EXPECT_EQ(arabic_to_roman(20), "XX");
        ...
        EXPECT_EQ(arabic_to_roman(90), "XC");
}
```

```
TEST(RomanTest, TestNegativeNumber) {
        EXPECT_EQ(arabic_to_roman(-1), "zzz");
}

TEST(RomanTest, TestOnesConversion) {
        EXPECT_EQ(arabic_to_roman(1), "I");
        EXPECT_EQ(arabic_to_roman(2), "II");
        EXPECT_EQ(arabic_to_roman(3), "III");
        EXPECT_EQ(arabic_to_roman(4), "IV");
        EXPECT_EQ(arabic_to_roman(5), "V");
        EXPECT_EQ(arabic_to_roman(6), "VI");
        EXPECT_EQ(arabic_to_roman(7), "VII");
        EXPECT_EQ(arabic_to_roman(8), "VIII");
        EXPECT_EQ(arabic_to_roman(9), "IX");
}

TEST(RomanTest, TestTensConversion) {
        EXPECT_EQ(arabic_to_roman(10), "X");
        EXPECT_EQ(arabic_to_roman(20), "XX");
        ...
        EXPECT_EQ(arabic_to_roman(90), "XC");
}

TEST(RomanTest, TestHundredsConversion) {
        EXPECT_EQ(arabic_to_roman(100), "C");
        EXPECT_EQ(arabic_to_roman(200), "CC");
        ...
        EXPECT_EQ(arabic_to_roman(900), "CM");
}
```

```
TEST(RomanTest, TestNegativeNumber) {
        EXPECT_EQ(arabic_to_roman(-1), "zzz");
}

TEST(RomanTest, TestOnesConversion) {
        EXPECT_EQ(arabic_to_roman(1), "I");
        EXPECT_EQ(arabic_to_roman(2), "II");
        EXPECT_EQ(arabic_to_roman(3), "III");
        EXPECT_EQ(arabic_to_roman(4), "IV");
        EXPECT_EQ(arabic_to_roman(5), "V");
        EXPECT_EQ(arabic_to_roman(6), "VI");
        EXPECT_EQ(arabic_to_roman(7), "VII");
        EXPECT_EQ(arabic_to_roman(8), "VIII");
        EXPECT_EQ(arabic_to_roman(9), "IX");
}

TEST(RomanTest, TestTensConversion) {
        EXPECT_EQ(arabic_to_roman(10), "X");
        EXPECT_EQ(arabic_to_roman(20), "XX");
        ...
        EXPECT_EQ(arabic_to_roman(90), "XC");
}

TEST(RomanTest, TestHundredsConversion) {
        EXPECT_EQ(arabic_to_roman(100), "C");
        EXPECT_EQ(arabic_to_roman(200), "CC");
        ...
        EXPECT_EQ(arabic_to_roman(900), "CM");
}


TEST(RomanTest, TestThousandsConversion) {
        EXPECT_EQ(arabic_to_roman(1000), "M");
        EXPECT_EQ(arabic_to_roman(2000), "MM");
        EXPECT_EQ(arabic_to_roman(3000), "MMM");
```

```
std::string arabic_to_roman(int arabicNumber)
{
  // Make sure number is positive.
  if ( arabicNumber <= 0 ) return "zzz";

  std::string result = "";

  // Split up the number into columns.
  const int ones = get_ones(arabicNumber);
  const int tens = get_tens(arabicNumber);
  const int hundreds = get_hundreds(arabicNumber);
  const int thousands_and_above = get_thousands_and_above(arabicNumber);

  // Register converted Roman numerals by column.
  const std::string roman_ones[10] = { "", "I", "II", "III", "IV",
                                       "V", "VI", "VII", "VIII", "IX" };
  const std::string roman_tens[10] = { "", "X", "XX", "XXX", "XL",
                                       "L", "LX", "LXX", "LXXX", "XC" };
  const std::string roman_hundreds[10] = { "", "C", "CC", "CCC", "CD",
                                           "D", "DC", "DCC", "DCCC", "CM" };

  // Recombine columns to form full Roman numeral.
  for (int i=0; i<thousands_and_above; i++) result.append("M");
  result.append(roman_hundreds[hundreds]);
  result.append(roman_tens[tens]);
  result.append(roman_ones[ones]);

  return ( result );
}
```

```
$> make; ctest --verbose
...
    Start 1: test_roman_test

1: Test command: /home/.../tests/test_roman
1: Test timeout computed to be: 9.99988e+06
1: Running main() from /home/.../src/gtest_main.cc
1: [==========] Running 7 tests from 1 test suite.
1: [----------] Global test environment set-up.
1: [----------] 7 tests from RomanTest
1: [ RUN      ] RomanTest.Dummy
1: [       OK ] RomanTest.Dummy (0 ms)
1: [ RUN      ] RomanTest.TestNegativeNumber
1: [       OK ] RomanTest.TestNegativeNumber (0 ms)
1: [ RUN      ] RomanTest.TestOnesConversion
1: [       OK ] RomanTest.TestOnesConversion (0 ms)
1: [ RUN      ] RomanTest.TestTensConversion
1: [       OK ] RomanTest.TestTensConversion (0 ms)
1: [ RUN      ] RomanTest.TestHundredsConversion
1: [       OK ] RomanTest.TestHundredsConversion (0 ms)
1: [ RUN      ] RomanTest.TestThousandsConversion
1: [       OK ] RomanTest.TestThousandsConversion (0 ms)
1: [ RUN      ] RomanTest.TestDiagonalsConversion
1: [       OK ] RomanTest.TestDiagonalsConversion (0 ms)
1: [----------] 7 tests from RomanTest (0 ms total)
1:
1: [----------] Global test environment tear-down
1: [==========] 7 tests from 1 test suite ran. (0 ms total)
1: [  PASSED  ] 7 tests.
1/1 Test #1: test_roman_test ..................   Passed    0.03 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.03 sec
```

Thanks to the organizers, lecturers, administrators, and fellow students for making this a great summer school!